(19) Japan Patent Office (JP)

(12) Official Patent Publication (A)

(11) Patent Application Publication HEI2-130635

(51) Int. Cl[4] | ID Office Ref. No. | (43) Publication May 18, 1990

G 06 F 9/38 | 370 A | 7361-5B

Examination Apply No apply | Number of claims 7 (total 12 pages)

(54) Title of the invention

System for simultaneously processing plural instructions

(21) Patent application SHO63-283679
(22) Application date November 11, 1988

(72) Inventor: Michio Morioka
c/o HITACHI SEISAKUJO, INC.
4026 Kuji-cho, Hitachi-shi, Ibaragi-ken

(72) Inventor: Kei-ichi Kurosawa
c/o HITACHI SEISAKUJO, INC.
4026 Kuji-cho, Hitachi-shi, Ibaragi-ken

(72) Inventor: Tada-aki Bando
c/o HITACHI SEISAKUJO, INC.
4026 Kuji-cho, Hitachi-shi, Ibaragi-ken

(72) Applicant: HITACHI SESAKUJO, INC.
6 Kanda-Surugadai 4-chome, Chiyoda-ku, Tokyo

(74) Attorney: Patent Attorney Katsuo Ogawa and two others

Specification

1. Title of the invention

System for simultaneously processing plural instructions

2. Claims

1. A system for simultaneously processing plural instructions in a multistage pipeline computer comprising an instruction reading unit for simultaneously reading multiple instructions from a main storage for storing instructions and operands, plural decoding units for interpreting said read instructions and identifying the types of said instructions and operands, plural operand reading units for reading required operands from said main storage or a general-purpose register file according to the results given by said decoding units, plural processing units for executing operations on said read operands according to the types of said instructions, wherein said decoding units comprise a decoding means for simultaneously decoding plural instructions and a determining means for determining whether said decoded instructions can be processed in parallel, and a means for coupling said instructions which can be processed in parallel, and wherein said plural instructions coupled are processed in pipelines synchronously between said plural operand reading means and plural processing units.

2. The system for simultaneously processing plural instructions according to Claim 1, characterized by the fact that said system comprises at least one status register for indicating the operation result status of said operations, wherein the operation result statuses of said plural instructions which are coupled and executed are reflected on said status register according to the sequence of said instructions.

3. The system for simultaneously processing plural instructions according to Claim 1, characterized by the fact that said system comprises at least one status register for indicating the operation result status and a means for transferring the operation result statuses

2

between plural pipelines, wherein it is determined whether a conditional branch instruction is executed according to a selection result between the operation result status which was transferred from another pipeline and the content of said status register.

4. The system for simultaneously processing plural instructions according to Claim 1, characterized by the fact that said system further comprises a means for transferring the operation results between plural pipelines, wherein the transferred operation results are used to execute the operations of other instructions.

5. The system for simultaneously processing plural instructions according to Claim 1, characterized by the fact that said system is performed in plural pipeline processing units consisting of the same logic hardware.

6. The system for simultaneously processing plural instructions according to Claim 1, characterized by the fact that said system further comprises a general-purpose register file shared by plural pipeline processing units and having plural read/write ports and a cache memory having plural read/write ports so that simultaneous processing is performed.

7. A system for simultaneously processing plural instructions in a multistage pipeline computer comprising an instruction reading unit for reading plural variable-length instructions from a main storage for storing variable-length instructions and operands, plural decoding units for interpreting said read instructions and identifying the types of instructions and the types of operands, plural operand reading units for reading required operands based on the decoding results from said decoding units, plural processing units for executing operations on said operands according to the types of the instructions, characterized by the fact that said decoding units comprise plural decoders consisted of the same number of bits as a minimum unit of variable-length instructions, and plural instructions which are read from said main storage are simultaneously decoded by using said decoders to simultaneously decode each minimum unit of

an instruction and then using the decoded results to identify the head of the instruction.

3. Detail explanation of the invention

[Scope of the invention]

The present invention relates to a processing unit which executes instructions sequentially, and componenticularly to a processing unit which executes plural instructions simultaneously. More precisely, the present invention relates to an architecture for executing plural instructions simultaneously while maintaining the succession of instructions in a processing unit consisting of plural pipelines.

[Prior art technology]

In the prior art, high performance of a general-purpose computer has been realized by multistage pipelining. Processes for executing an instruction, such as instruction fetching, decoding, operand address calculation, operand fetching, operation, are considered to be distinct stages. Different instructions are executed at each stage so that high performance is realized. It is assumed in a multistage pipeline system that instructions are executed sequentially. In other words, instructions are executed sequentially based on a program counter and their execution sequence is never changed unless program control instructions such as branch instructions are executed. Instructions are executed in a conventional general-purpose computer, assuming the succession of instruction execution is maintained.

On the other hand, attempts have been made to speed up the process by executing plural instructions in parallel for a long time. These include, for instance, CDC5500 described in "Parallel Operation in the Control Data 6600", Proc. of Spring Joint Computer Conference, 1964 and, more recently, Motorola's MC88100 described in "Supercomputing on Chip" VLSI System Design May 1988, pp24 - 33. Similar architecture can be seen in "Multi

execution unit, uni-processor system" described in Patent Publication No. SHO62-262142.
In CDC500 and MC88100, fixed-point and floating-point calculations are restricted to data between general-purpose registers. Data transfer between general-purpose registers and the main memory is executed by dedicated load/store instructions. Plural processing units are provided and they can operate independently. This data structure allows the parallel execution of a transfer instruction between the main memory and general-purpose registers and an operation instruction, as well as plural operation instructions. This architecture executes transfer instructions and operation instructions asynchronously. Asynchronous execution of these instructions is useful for deriving potential parallel execution in a program. However, there are some problems, too.

The first problem is that a complex control mechanism may be required to maintain the instruction sequence. For executing an operation on a datum, the datum in question is transferred from main memory to a general-purpose register and the operation is executed on the datum in the general-purpose register, the result being transferred from the general-purpose register to the main memory. This process is executed with three instructions, i.e. load, operation, and store instructions and must be executed sequentially. However, the sequence of the instructions is not assured if they are executed asynchronously. Therefore, the CDC5500 architecture uses a scoreboard system to maintain the sequence of instructions. The scoreboard system employs flags for exclusive control of each entry in the general-purpose register, known as a scoreboard bit. When an instruction is decoded, the flag in the general-purpose register which includes the operand of the instruction is turned on. The flag is cleared when the execution of the instruction is completed. An instruction which attempts to access a register whose flag is ON is blocked and kept waiting until the flag is turned OFF. This ensures maintaining the sequence of instructions described above. In the "plural execution unit, uni processor system" described in Patent Publication No. SHO 62-263142, plural processing units

can be provided to execute plural instructions asynchronously. Its architecture comprises a more extensive exclusive control mechanism for general-purpose registers in order to ensure the sequence of instructions. This exclusive control mechanism for general-purpose registers increases hardware complexity and may cause a reduction in the performance of operation processing. Conventional pipeline systems ensures the sequence of instructions when a series of sequential processes, such as load, operation, and store, are executed. Therefore, it is possible to simplify exclusive control for general-purpose resisters and to transfer loaded data directly to operation instructions without passing through general-purpose registers. On the other hand, the CDC5500 architecture stores the loaded data in general-purpose registers and operation instructions are held until the score board flag is cleared. This causes a delay in transferring data between the instructions due to the exclusive control overhead.

The second problem accompanying asynchronous execution of instructions complicates the status control of the processing units. Conventional single pipeline systems do not change the execution sequence of instructions. The status of processing units changes according to the order of instructions, reflecting the status registers. As long as this is assured, status control is easy. For instance, it assures that the execution results of the preceding instructions are stored in status registers before a condition is determined for a conditional branch instruction which transfers control according to the status of the processing units. For interruptions, the status of the processing units at the time of interruption can be easily determined. It is also easy to reproduce the status of processing units when interruption occurs after some procedures are executed for n interrupt request.

Conversely, when instructions are executed asynchronously, there is no guarantee that the instructions are executed sequentially, complicating the status management of processing units. For instance, if an instructions for generating a condition and a

conditional branch instruction are asynchronously executed, there is no assurance that the execution result to generate the condition is reflected in the status register before the conditional branch instruction is executed. To solve this problem, the Motorola's MC88100 uses a system for transferring the execution result status of an instruction to generate a condition to a conditional branch instruction via a general-purpose register. This system allows synchronizing an instruction to generate a condition with a conditional branch instruction by means of the exclusive control mechanism of general-purpose registers. However, this system requires conditional branch instructions to have a format in which they can specify a general-purpose register with its operand. This system is unavailable in a computer which executes the instruction set without this format.

[Problems overcome by the invention]

As is described above, a conventional system for simultaneously processing plural instructions wherein instructions are executed asynchronously using plural different execution units has the problem that a complex exclusive control mechanism is required for general-purpose registers in order to ensure the essentially sequential processing of instructions. In addition, the exclusive control mechanism may cause a reduction in performance due to the data transfer overhead among instructions. Furthermore, the management of the status registers of processing units is complicated because asynchronously execution of instructions does not ensure their sequence.

Patent Publication No. SH062-65133 is known for a means for executing plural instructions simultaneously within the time for executing an instruction. However, no description is provided on how plural instructions are to be executed.

A purpose of the present invention is to provide a processing unit and a system for simultaneously processing plural instructions, which enables simultaneous processing of plural instructions while

maintaining their succession according to the sequence of instructions stated in a program.

Another purpose of the present instruction is to provide a system for simultaneously processing plural instructions, comprising at least one status register for plural processing units, said status register being updated according to the sequence of instructions stated in a program.

Another purpose of the present invention is to provide a system for simultaneously processing instructions, wherein the general-purpose register file does not require an exclusive control mechanism.

Another purpose of the present invention is to provide a system for simultaneously processing instructions, comprising repetitive hardware structures having the same logic.

[Problem resolution means]

The above purposes are attained by configuring a processor using plural pipelines consisting of hardware of the same logic and by a system for simultaneously processing plural instructions, comprising a means for decoding plural instructions simultaneously, a means for determining whether the decoded instructions can be executed in parallel and, if so, coupling these instructions, and a means for always executing the coupled instructions synchronously in plural pipeline processing units. A processor consisting of plural pipelines is provided in the sole status register which indicates an operation result status. The operation result statuses of the plural instructions coupled are combined according to the instruction sequence in a program, and simultaneously reflected in the status register. The register file and cache memory are shared by plural pipelines, having plural read/write ports for providing operands simultaneously for each pipeline.

[Efficacy]

The decoder decodes, picks up, and analyses plural instructions simultaneously, regardless of whether they are fixed-length or variable-length instructions. It is determined that the plural instructions picked up can be executed in parallel by analyzing the operand combination or comparing the instruction types. If the parallel execution is available, these instructions are coupled and executed using plural pipelines. In this case, the plural instructions coupled are executed synchronously. In other words, these plural instructions are present at the same stage of each pipeline regardless the complexity of the instruction. The operation result statuses in the pipelines are merged according to the instruction order and reflected in the sole status register. Thus, plural instructions can be executed without their sequence being changed while maintaining their succession. Ensuring the succession of instructions allows simplification of the exclusive control of the general-purpose registers. It is also ensured that the sole status register is updated according to the instruction sequence, facilitating the status management of processing units for conditional branch instructions and interruptions.

[Embodiments]

An embodiment of the present invention is described hereinafter, with reference to the drawings. Fig.2 shows an example of a computer system to which the present invention is applied. Cluster computers 100, 110, and 120 are connected to a global memory 130 at global memory ports 131, 132, and 133. The cluster computers share the global memory 130 which is duplicated for high reliability. Each cluster computer is connected to magnetic disks 141, 142 or terminals 143, 144 via an I/O switching network 140. In the cluster computer 100, processing units 103, 104, 105, and 106 are connected to a common memory 101 via a common bus 102 and a memory port 108. In the common memory 101 are stored programs and data necessary for the processing units. An input/output port 107 is used for the processing units to access input/output devices such as the magnetic disks 141 and 142.

An internal architecture of the processing unit 103 is described hereafter, with reference to Fig.3. An instruction cache memory 230 temporally holds the instructions to be executed in the processing unit 103. An instruction fetch unit 200 reads the instructions from the instruction cache memory 230 and transfers them to the instruction execute unit 210. Logical addresses 201 which are output from the instruction fetch unit 200 are converted to physical addresses by an instruction address convert buffer 220 and supplied to the instruction cache memory 230. Instructions read from the instruction cache memory 230 are supplied to the instruction fetch unit 200 via a bus 202. Other functions of the instruction fetch unit 200 include the control of instruction fetch direction, for which it has an internal branch predict buffer. When the instruction fetch unit 200 detects a branch instruction among the fetched instructions, it accesses the branch predict buffer to determine the instruction address to branch. An operand cache memory 250 temporarily holds operands accessed by an instruction execute unity 210. An operand address convert buffer 240 converts logical addresses 203 which are output from the instruction execute unit 210 to physical addresses and transfers them to the operand cache memory 250. The instruction execute unit receives instructions from the instruction fetch unit 200, decodes them, and performs operand address calculation, operand fetch, and operation according to the decoding results. A common bus monitor 280 monitors transactions on a common bus 102 and, if necessary, cancels or updates the operand cache memory 250. This ensures the consistency of the operand cache memories provided in plural processing units.

The instruction fetch unit 200 is described next, with reference to Fig.4. A fetch pointer 300 holds the addresses of instructions to fetch. A selector 302 selects an adder 301 and the fetch pointer is increased by a certain increment as far as instructions are fetched sequentially. In this embodiment, 16 byte length data is read per each instruction fetch and accordingly the fetch pointer 300 is increased by 16. If a branch instruction is included in the fetched instructions, the selector 302 selects a branch address

10

304 which is sent from the branch predict buffer 330 or the instruction execute unit, and sets the fetch pointer 300 at the branch address. An instruction is read from the instruction cache memory 230 according to the address in the fetch pointer 300 and stored in the instruction buffer 310. The instruction buffer 310 is a first-in first-out buffer and is assumed to have eight 16-byte entries. The instruction buffer 310 has a read address register 312. The read address register 312 indicates any byte position of the instruction buffer 310. An aligner 311 reads 18-byte information from the indicated byte position and sends it to a decoder 314. An instruction pick up component 315 informs an adder 313 of the size of the picked up instruction and a new value is determined for of a read address register 312. The decoder component 314 decodes the 16-byte information read from the instruction buffer using plural decoders having the same number of bits as the minimum unit of instructions. Here, the minimum unit of instructions is 2 bytes and accordingly 16-byte information is decoded simultaneously using eight decoders for two bytes each. The analysis results from these eight decoders are transferred to an instruction pick up 315. The instruction pick up 315 picks up a first instruction 319 and determine the size 316 of the first instruction as well as picks up a second instruction 325 and determines the size 326 of the second instruction size 326 according to the information given by the decoder. In this embodiment, two instructions are simultaneously picked up. However, more than two instructions can be simultaneously picked up. With the decoding system described above, plural variable length instructions can be simultaneously picked up. The picked up first and second instructions 319 and 325 and their size information 316 and 317 are simultaneously stored in the execute unit instruction buffer 340.

On the other hand, a program counter 320 holds the address in the main memory of the first instruction picked up by the decoder 314. An adder 323 is used to calculate the address of the second instruction, which adds the size 316 of the first instruction to the program counter 320. The addresses in the main memory

of the first and second instructions are additionally stored when the instructions are stored in the execute unit instruction buffer 340. The program counter 320 is updated to a new value by an adder 321 which calculates the sum of the sizes of the first and second instructions picked up by the instruction pick up 315 unless a branch instruction is included. When a branch instruction changes the program flow, the program counter 320 is set at what a selector 322 selects, either the predicted address from the branch predict buffer 330 or the branch address 325 from the instruction execute unit.

The branch predict buffer 330 is described next. The information which is stored in the branch predict buffer 330 includes the following six items.

1) Validity bid 331 ... indicating each entry is valid.

2) comparison address tag 332 ... holding component of the address of a branch instruction compared with the address obtained from an external source using a comparator 336 to verify that the branch instruction in question is present in the branch predict buffer 330.

3) branch predict bit 333 ... if the branch instruction in question is a conditional branch instruction, information is provided as to whether the branching should be performed or not.

4) branched instruction address 334 ... the address of the branched instruction to which the branch instruction in question predictably jumps.

5) branched instruction 335 ... the instruction itself to which the branch instruction in question predictably jumps.

The branch predict buffer 330 stores the history of the executed branch instructions and predicts the branch address when the same branch instruction appears again. It operates as follows. The instruction pick up 315 picks up instructions and, if a branch instruction is

included, it sends the address of the branch instruction to the branch predict buffer 330 by controlling the selector 324 based on which instruction it is, the first or second. The branch predict buffer selects a specific entry of the branch predict buffer using the address sent and compares it with the compare address tag 332 using the comparator 336 to determine whether the branch instruction in question has been registered. If the branch instruction is registered and the branch predict bit 333 indicates a branch, the branch address is set in the program counter 320 and fetch pointer 300. At this time, the instruction buffer 310 is entirely cleared. Then, the branched instruction 335 is stored in the instruction buffer 310 using the selector 303. On the other hand, no operation is performed if the branch predict bit 333 indicates no branch.

The instruction execute unit 210 is described in detail hereafter, with reference to Fig.5. The embodiment in Fig.5 is configured to execute two instructions simultaneously, and can be easily configured to execute more than two instructions simultaneously, as well. An execute unit instruction buffer 340 reads two instructions simultaneously and decoders 400 and 401 determine the instruction type and operand type. In this instance, it is assumed that the decoder 400 decodes an instruction to be executed first (first instruction) and the decoder 401 decodes an instruction to be executed next (second instruction). The information is sent to an instruction coupling checker 402. The instruction coupling checker verifies the instruction type and the operand conflict and determines whether it is possible to couple the two instructions read from the execute unit instruction buffer 340. Possible coupling of instruction types are shown in Fig.7. Almost all instruction pairs can be coupled except for bit field and decimal operation instructions not coupled to other instructions. In addition, branch instructions and subroutine link instructions cannot be coupled with themselves. Furthermore, two instructions cannot be coupled when the destination operand of one instruction is equal to the source operand of the other instruction.

For the instruction pair which is determined to be coupled by the instruction coupling checker 402, the

following pipeline stages such as address calculation, operand fetch, and operation are synchronously executed. For the pair determined not to be coupled, only the first instruction is given to the following stages. The remaining instruction is treated as the first instruction in the following pair and decoded and determined together with the following instruction whether they can be coupled.

The decoding results of the coupled instructions are set in groups of registers 410 to 414 for the first instruction and 415 to 419 for the second instruction. The register operand address register 410 and 411 store the register addresses of source and destination operands of the first instruction. The registers 418 and 419 serve the same function for the second instruction. When the first instruction includes a memory operand, the register address of the base register is stored in the register 414, the register address of the index register in the register 413, and the displacement information in the register 412. The registers 415, 416, and 417 serve the same function for the second instruction.

The process of the address calculation stage is described. When the first instruction includes a memory operand, its logical address has to be calculated. The address of the memory operand is calculated using an adder 421, which adds the contents of the base register in the address register file 420 indicated by the register 414, the contents of the index register in the address register file 420 indicated by the register 413, and the displacement information 412. The calculated address is stored in a logical address register 425. The same procedure is performed for the second instruction and the calculated logical address is stored in a register 426. The address register file 420 is shared by the first and second instructions, having plural read ports to allow simultaneous address calculation for the first and second instructions.

The memory operand read stage is described. When the first operand includes a memory operand, the memory is accessed using the logical address 425 obtained in the address calculation stage. The logical address 425 is

converted to a physical address in the main memory by an operand address convert buffer 430. Using the physical address, an operand cache memory 431 is accessed and the read memory operand is stored in a register 434. In the same way, the memory operand is stored in a register 435 for the second instruction. The operand address convert buffer 430 and the operand cache memory 431 are shared by the first and second instruction, having plural read port to allow the simultaneous operand reading for the first and second instructions. It is noted here that, for instance, if a cache miss occurs in the first instruction, it cannot proceed to the following stage, the second instruction can not proceed to the following stage.

The configuration of operand fetch stage is described. If the first instruction has a register operand, the operand is read from a data register file 440 according to the information in register operand address registers 432 and 433. On the other hand, if it is a memory operand, the operand is obtained from a register 434 via an aligner 441.

If the source operand is the result of an instruction executed at in the immediately preceding step in the same pipeline, the operand is obtained from an intra-pipe bypass rout 460. On the other hand, the source operand is the result of instructions executed in the immediately preceding step in another pipeline, and the operand is obtained from an inter-pipe bypass route 461. The same procedures are performed for the second instruction.

Processing units 454 and 455 execute operations on the operands obtained at the operand fetch stage and the results are stored in registers 456 and 457. Following this, the operation results are stored in the address register file 420, data register file 440, or operand cache memory 431. The operation results statuses 462 and 463 (zero, overflow, and so on) of the processing units 454 and 455 are transferred to a status code generation circuit 458 and reflected on a status register 459.

The status code generation circuit is described hereafter, with reference to Fig.5. The status code generation circuit 458 has two functions. The first function is to

merge the operation result statuses of the first and second instructions, considering the instruction sequence and reflects it on the status register. The second function is to determine the condition for simultaneously processing a conditional branch instruction and an instruction to generate the condition. In the first function, the operation result status 462 which is output from the processing unit 454 for the first instruction and the operation result status 463 which is output from the processing unit 455 for the second instruction are input into a status generation component 916. Considering that the first instruction should be executed before the second instruction, the status generation component 916 reflects the status from the second instruction on the status 462 from the first instruction and stores it in the status register 459.

The second function is as follows. Assuming that a conditional branch instruction is executed in the pipeline for the second instruction, the branch condition determining information is stored in a register 904 and prediction of the results by the branch predict buffer is stored in a register 905. Assuming that an instruction to generate a branch condition and a conditional branch instruction are sequentially executed, the branch condition is already reflected on the status register when the conditional branch is executed. Therefore, a selector 914 selects the status register 459 and inputs it in a branch determining circuit 915. The branch determining circuit 915 determines whether the branch should be performed using the status register 459 and the branch condition determining information 904, the result of which is compared with the branch prediction results 905 using a comparator 916. If it is found to be consistent, no operation is performed. If not consistent, all of the pipelines are canceled to branch correctly. Now, a case is considered in which an instruction for generating a condition and a conditional branch instruction are executed simultaneously. It is assumed that the first instruction is an instruction for generating a condition and the second instruction is a conditional branch instruction. In this case, the branch condition is not reflected on the status register 469

16

when the conditional branch instruction is executed. Therefore, the selector 914 selects the operation result status 462 of the first instruction and inputs it into the branch determining circuit 915. The branch determining circuit 915 determines whether the branch should be performed using the operation results status 462 of the first instruction and the branch condition determining information 904, the results of which are compared with the branch predict result 905 using the comparator 916. If it is found to be consistent, no operation is performed. If not consistent, all of the pipelines are canceled to branch correctly.

The configuration of the status code generation circuit 456 described above ensures that the status register 459 is updated according to the instruction sequence and allows the simultaneous execution of a conditional branch instruction and an instruction for generating a branch condition.

The operation of the pipelines is described hereafter, with reference to Figs. 1, 8, and 9. Fig.1 shows the pipeline configuration of an embodiment of the present invention which realizes the simultaneous processing of two instructions. Plural instructions are read from an instruction cache memory 520 at the instruction fetch stage 500. The plural instructions are simultaneously picked up by the pre-decode stage 501. If a branch instruction is included, the branch direction is determined by accessing a branch predict buffer 521. Instructions are read from an execute unit instruction buffer in the instruction buffer state 502. Two instructions are simultaneously decoded and it is determined whether they can be coupled at the decode and combine stage 503. The logical addresses of the memory operands are calculated at the address calculation stages 504 and 511 and they are converted to logical addresses at the address convert stages 506 and 512. The operands are read from an operand cache memory 523 or a register file 522 at the operand fetch stages 505 and 515. Operations are performed on the read operands by the operation stage 509 and 516. The operation results are stored in the operand cache memory 523 or register file 522 by the write stage 510 and 517. After the address calculation stage, two pipelines have the same logic.

The coupled instructions at the decode and combine stage 503 are synchronously executed in two pipelines.

Fig.8 shows the pipeline stage flowchart in which the simultaneous processing of two instructions is effectively performed. In the figure, simultaneous processing of the third and fourth instructions is performed by transferring the operation result between the pipelines, as described above. Simultaneous processing of the ninth and tenth instructions is performed when the branch is successfully predicted for the subroutine jump instruction of the ninth instruction.

Fig.9 shows another a pipeline stage flowchart for the simultaneous processing of two instructions. This shows an example in which the third and fourth instructions cannot be coupled because of a d7 register conflict. In this case, only the third instruction is executed independently and the fourth instruction is coupled with the fifth instruction to execute them. The eighth and ninth instructions are successfully coupled. However, the a0 register conflicts between the seventh and eighth instructions, so that the eighth instruction must wait. In this case, the ninth instruction which is coupled to the eighth also must wait. Thus, synchronous pipelines ensure that the sequence of instructions is maintained.

[Efficacy of the invention]

The present invention enables simultaneous processing of plural instructions while maintaining their sequence, simplifying the exclusive control of general-purpose register file and allowing high performance. This ensures that the sole status register is updated according to the instruction sequence, facilitating the status management of processing units.

4. Brief description of the drawings

Fig.1 is a schematic presentation showing the pipeline configuration of an embodiment of the present invention. Fig. 2 a schematic presentation showing a computer system to which the present invention is applied. Fig.3 shows the internal configuration of the processing unit

18

in Fig.2. Fig.4 shows the internal configuration of the instruction fetch unit in Fig.3. Fig.5 shows the internal configuration of the instruction execute unit in Fig.3. Fig.6 shows the internal configuration of the status code generation circuit in Fig.5. Fig.7 shows the possible coupling of instructions. Figs. 8 and 9 shows examples of the pipeline stage flowchart.

500 … instruction fetch stage,
503 … decode and combine stage,
504, 511 … address calculation stage,
508, 515 … operand fetch stage,
509, 516 … operation stage,
522 … multi-port register file,
523 … operand cache.

Agent Patent Attorney Katsuo Ogawa

⑲日本国特許庁(JP)　⑪特許出願公開

⑫公開特許公報(A)　平2−130635

⑤1Int.Cl.⁵	識別記号	庁内整理番号	⑬公開　平成2年(1990)5月18日
G 06 F 9/38	370 A	7361−5B	

審査請求　未請求　請求項の数 7　(全12頁)

⑤4発明の名称　複数命令同時処理方式

⑳特　願　昭63−283679
㉒出　願　昭63(1988)11月11日

⑦2発明者　森岡　道雄　茨城県日立市久慈町4026番地　株式会社日立製作所日立研究所内
⑦2発明者　黒沢　憲一　茨城県日立市久慈町4026番地　株式会社日立製作所日立研究所内
⑦2発明者　坂東　忠秋　茨城県日立市久慈町4026番地　株式会社日立製作所日立研究所内
⑦1出願人　株式会社日立製作所　東京都千代田区神田駿河台4丁目6番地
⑦4代理人　弁理士　小川　勝男　外2名

明　細　書

1．発明の名称

複数命令同時処理方式

2．特許請求の範囲

1．命令およびオペランドを蓄積する主記憶装置から、複数個の命令を同時に読出す命令読出し装置と、読出された命令を解読し、命令の種類とオペランドの種類を識別する複数個のデコード装置と、デコード装置の結果に基づいて、必要なオペランドを、前記主記憶装置、あるいは、汎用レジスタファイルから読出す複数のオペランド読出し装置と、読出されたオペランドに対して、命令の種類に従い、演算を行なう複数の演算装置から成る多段パイプライン計算機において、該デコード装置は複数命令を同時にデコードするデコード手段と、デコードした複数の命令が並列に実行可能かどうかを識別する識別手段と、並列実行可能な命令を結合する手段を有し、結合された複数の命令を、複数のオペランド読み出し手段および複数の演算装置間で同期してパイプライン処理するようにしたことを特徴とする複数命令同時処理方式。

2．請求項1記載の複数命令同時処理方式は、演算処理結果の状態を示す少なくとも1つの状態レジスタを有し、結合して実行される複数命令の演算結果の状態を、命令の順序に従って、状態レジスタに反映するようにしたことを特徴とする複数命令同時処理方式。

3．請求項1記載の複数命令同時処理方式は、演算処理結果の状態を示す少なくとも1つの状態レジスタと、複数のパイプライン間で、演算結果の状態を転送する手段と、他パイプラインから転送されてきた演算結果の状態と、前記状態レジスタの内容のどちらかを選択し、選択された結果に基づいて条件分岐命令を実行するか否かを決定するようにしたことを特徴とする複数命令同時処理方式。

4．請求項1記載の複数命令同時処理方式は、複数のパイプライン間で、演算結果を転送する手段を有し、転送された演算結果を用いて、他の

命令の演算を行なうようにしたことを特徴とする複数命令同時処理方式。

5．請求項1記載の複数命令同時処理方式は、同一論理のハードウエアから成る複数のパイプライン処理装置によつて処理されることを特徴とする複数命令同時処理方式。

6．請求項1記載の複数命令同時処理方式は、複数のパイプライン処理装置によつて共有された、複数の読出し／書込みポートを持つ汎用レジスタファイルと、複数の読出し／書込みポートを持つキヤツシユメモリを有し、同時処理されることを特徴とする複数命令同時処理方式。

7．可変長命令およびオペランドを蓄積する主記憶装置から複数個の可変長命令を同時に読出す命令読出し装置と、読出された命令を解読し、命令の種類とオペランドの種類を識別する複数個のデコード装置と、デコード装置のデコード結果に基づいて、必要なオペランドを読出す複数のオペランド読出し装置と、該オペランドに対して、命令の種類に従い演算を行なう複数の演算装置から成る多段パイプライン計算機において、前記デコード装置は可変長命令の最小単位と等しいビット幅から成る複数のデコーダを有し、該デコーダは主記憶装置より読出された複数の命令に対して該命令の最小の単位毎に同時にデコードし、デコードした結果に基づいて各命令の先頭を識別し、複数の命令を同時にデコードするようにしたことを特徴とする複数命令同時処理方式。

3．発明の詳細な説明

〔産業上の利用分野〕

本発明は、逐次的に命令を実行する演算処理装置に係り、特に複数の命令を同時に実行する演算処理装置に関する。更に詳しくは、本発明は、複数のパイプラインからなる演算処理装置において、命令の逐次性を維持しつつ、複数の命令を同時に実行するための，アーキテクチヤに関する。

〔従来の技術〕

従来、汎用計算機の高性能化は、パイプラインを多段化することによつて実現されてきた。これは、1つの命令の実行に必要な処理、例えば、命令フエツチ，デコード，オペランドアドレス計算，オペランドフエツチ，演算等の処理を、それぞれ独立なステージとし、ステージ間で異なる命令を実行することにより高性能化を図る手段である。多段パイプライン方式では、命令が逐次的に実行されることを前提としている。すなわち、分岐命令等のプログラム制御命令が実行されない限り、プログラムカウンタに基づいて、逐次的に命令が実行され、命令の実行順序が入れ変わることは、あり得ない。従来の汎用計算機で実行される命令は、この命令実行の逐次性を前提とした仕様となつている。

一方、単一のプロセツサにおいて、複数命令を並列に実行することによつて高速化を図る試みが古くからなされてきた。例えば、“Parallel Operation in the Control Data 6600,” Proc.of Spring Joint Computer Conference, 1964に記載されているCDC6600、最近の計算機では、“Supercomputing on Chip,” VLSI Systems Design may 1988,pp２４－３３に記載されているモトローラ社のMC88100 等がある。又、特開昭62－262142号に記載の“複数実行ユニツト・ユニプロセツサシステム”も同様のアーキテクチヤと考えられる。

CDC6600，MC88100は、固定小数点，浮動小数点の演算を、汎用レジスタ間のデータにのみ限定している。汎用レジスタと、主メモリ間のデータ転送は、専用のロード／ストア命令によつて実行される。又、演算ユニツトは複数個設けられ、独立に動作可能となつている。この様なデータストラクチヤを採用することにより、主メモリと、汎用レジスタ間の転送命令と演算命令、あるいは、複数の演算命令を並列に実行することが可能となる。本アーキテクチヤにおいては、転送命令，演算命令は非同期に実行される。この命令実行の非同期性は、プログラム中に内在する並列性を引き出すという点では有効であるが、いくつかの問題点を含む。

第1の問題点としては、命令の逐次性を維持す

るために複雑な制御機構が必要となることである。すなわち、あるデータに対して演算を施す場合、対象となるデータを主メモリから汎用レジスタに転送し、汎用レジスタ上のデータに対して演算を行ない、その結果を汎用レジスタから主メモリへ転送する。これらの処理は、ロード命令，演算命令，ストア命令の３つの命令によつて実現され、且つ、これらの３命令は、逐次的に実行されなければならない。しかし、もし、各命令が非同時に実行されるならば、命令の逐次性を保証できない。このため、CDC6600 アーキテクチヤでは、スコアボード方式を採用することにより命令の逐次性を維持している。これは、汎用レジスタに各エントリーにスコアボードビツトと呼ばれる排他制御用のフラグを設ける方式である。命令をデコードした時点で、該命令のオペランドを含む汎用レジスタのフラグをＯＮにする。該命令の実行が完了した時点で、ＯＮしていたフラグをクリアする。スコアボードビツトがＯＮとなつているレジスタにアクセスしようとした命令は、ブロツクされフラグＯＦＦとなるまでアクセスできない。これによつて、前述した命令の逐次性を維持する。特開昭62－262142号に記載の“複数実行ユニツト・ユニプロセツサシステム”においても、複数の演算ユニツトが設けられ、非同期に複数の命令が実行され得る。本アーキテクチヤにおいても、命令の逐次性を保証するために、汎用レジスタに、より拡張された排他制御機構を設けている。この様な汎用レジスタファイルの排他制御機構は、ハードウエアの複雑さが増すことに加え、演算処理の性能低下の原因にもなり得る。すなわち、ロード，演算，ストアといつた一連の逐次的な処理を実行する場合、従来の単一パイプライン方式では、命令の逐次性が保証されているため、汎用レジスタの排他制御を簡略化することが可能であり、ロードしたデータを汎用レジスタを介さず直接演算命令に渡すことが可能であつた。一方、CDC6600 アーキテクチヤでは、ロードしたデータが汎用レジスタに格納され、スコアボードフラグがクリアされるまで、次の演算命令を実行できない。すなわち、排他制御のオーバヘツドにより、命令間のデータ受渡しが遅れることになる。

又、命令実行の非同期性に伴なう第２の問題点としては、演算処理装置の状態の管理の複雑さを挙げることができる。従来の単一パイプライン方式では、命令の実行順序が変化することがなく、命令の順序に従つて、演算処理装置の状態が変化しこれが状態レジスタに反映される。これが守られるならば、状態の管理は容易である。例えば、演算処理装置の状態に従つて分岐する条件分岐命令を実行する場合、条件の判定を行なう時には、それ以前の命令の実行結果が、状態レジスタに反映されていることが保証される。又、例えば、割込要求があつた場合、割込要求が発生した時点の演算処理装置の状態は容易に判別可能である。そして、割込要求に対する何らかの処理を実施した後、割込んだ時点の演算処理装置の状態を再現することも容易である。

これに対し、非同期に命令が実行される場合には、命令が逐次的に実行される保証がなく、演算処理装置の状態の管理が複雑になる。例えば、前述の条件分岐命令と、条件を生成する命令が非同時に実行されるならば、条件分岐命令を実行する時に、条件を生成する命令の実行結果が状態レジスタに反映されているとは限らない。この問題に対して、モトローラ社のMC88100 では、条件生成命令の実行結果の状態を汎用レジスタを介して条件分岐命令に渡す方法を採つている。この方法によれば、汎用レジスタの排他制御機構により、条件生成命令と条件分岐命令の同期をとることが可能となる。しかし、本方式では、条件分岐命令が、そのオペランドとして汎用レジスタを指定できる命令仕様となつていることが不可欠である。該命令仕様を持たない命令セツトを実行する計算機では、本方式は実現できない。

〔発明が解決しようとする課題〕

上記の様に、複数の異なる実行ユニツトを設け非同期に命令を実行する様な従来の複数命令同時処理方式では、本質的に逐次的な処理の実行順序を保証するために、汎用レジスタにおいて複雑な

排他制御機構が不可欠となる問題があつた。又、該排他制御機構に起因して、命令間のデータの受渡しのオーバヘッドが大きくなり性能が低下する問題があつた。更には、命令が非同期に実行されるため、命令が順序に従つて実行されるとは限らず演算処理装置の状態レジスタの管理が複雑になる問題があつた。

複数命令を1命令実行時間に同時に実行させる例として特開昭62－65133 号公報が知られているが、ここには、複数命令を具体的にどのように実行するかが開示されていない。

本発明の目的は、プログラムに記述された命令の順序に従い、逐次性を維持しつつ複数命令を同時に処理可能な演算処理装置および複数命令同時処理方式を提供することにある。

本発明の他の目的は、複数の演算処理装置に対して少なくとも1つの状態レジスタを持ち、該状態レジスタが、プログラム中に記述された命令の順序に従い更新されることを保証する複数命令同時処理方式を提供することにある。

本発明の他の目的は、汎用レジスタファイルにおいて、排他制御機構を不要とする複数命令同時処理方式を提供することにある。

本発明の他の目的は、同一論理のハードウエアを繰返し用いて構成することが可能な複数命令同時処理方式を提供することにある。

〔課題を解決するための手段〕

上記目的は、同一論理のハードウエアからなる複数のパイプラインによつてプロセツサを構成し、複数命令を同時にデコードする手段と、デコードした複数の命令が並列に実行可能かどうか識別し可能ならば、これら複数の命令を結合する手段と結合された複数の命令を、複数のパイプライン処理装置で常に同期させて実行する手段から成る複数命令同時処理方時によつて達成される。複数のパイプラインから構成されるプロセツサには、演算処理結果の状態を示す唯一の状態レジスタが設けられる。結合された複数命令の演算結果の状態は、プログラム中の命令の順序に従つて結合され同時に状態レジスタに反映させる。又、レジスタファイル及び、キヤツシユメモリは、複数のパイプラインによつて共有され、各パイプに対して、同時にオペランドを供給できる様に複数のリードライトポートを持つ。

〔作用〕

デコーダは、対象となる命令が固定長命令、あるいは可変長命令に係わらず、同時に複数の命令をデコードし、命令の切出し及び、分析を行なう。切出された複数の命令は、オペランドの競合検証あるいは、命令種類の比較が行なわれ、並列実行可能かどうかの判定が行なわれる。並列実行可能であれば、これらの命令は結合され、複数のパイプラインによつて実行される。この時、結合された複数の命令は同期して実行される。すなわち、命令の複雑さに係わらず結合された複数の命令は各パイプラインの同一のステージに存在する。又、各パイプラインにおける演算結果の状態は、命令の順序に従つてマージされ、唯一の状態レジスタに反映される。これによつて、命令の実行順序を変えることなく、逐次性を維持して複数の命令を同時に処理することが可能となる。命令の逐次性が保証されることによつて汎用レジスタにおける排他制御を簡略化することが可能となる。又、唯一の状態レジスタは、命令の順序に従つて更新されることが保証され、条件分岐命令、あるいは割込処理における演算処理装置の状態の管理が容易となる。

〔実施例〕

以下、本発明の一実施例を図面を用いて説明する。第2図は、本発明が適用される計算機システムの1例を示している。クラスタコンピュータ100，110，120は、それぞれグローバルメモリポート131，132，133によつてグローバルメモリ130に接続される。各クラスタコンピュータは、グローバルメモリ130を共有しており、グローバルメモリは高信頼化のため2重化される。又、各クラスタコンピュータは、I／Oスイツチングネツトワーク140を介して磁気ディスク141，142、あるいは、端末装置143，144に接続される。クラスタコンピュ

ータ１００の内部では、演算処理装置１０３，１０４，１０５，１０６が、共有バス１０２、メモリポート１０８を介して、共有メモリ１０１に接続される。共有メモリ１０１には、各演算処理装置で必要となるプログラムや、データが格納される。演算処理装置から磁気ディスク１４１，１４２等の入出力装置をアクセスする場合は、入出力ポート１０７を介して行なわれる。

次に、第３図を用いて、演算処理装置１０３の内部構成について詳細に説明する。命令用キャッシュメモリ２３０は、演算処理装置１０３にて実行する命令を一時的に保持する。命令フェッチユニット２００は、命令用キャッシュメモリ２０３より命令を読出し、命令実行ユニット２１０に転送する。命令フェッチユニット２００の送出する論理アドレス２０１は、命令用アドレス変換バッファ２２０によつて物理アドレスに変換され、命令用キャッシュメモリ２３０に供給される。命令用キャッシュメモリ２３０より読出された命令はバス２０２を介して命令フェッチユニット２００に供給される。命令フェッチユニット２００の他の機能としては、分岐予測バッファを内蔵しており、フェッチした命令の中に分岐命令を検出すると、分岐予測バッファをアクセスすることにより、分岐先の命令アドレスを識別し、命令フェッチの方向制御を行なう。オペランド用キャッシュメモリ２５０は、命令実行ユニット２１０によつてアクセスされるオペランドを一時的に保持する。オペランド用アドレス変換バッファ２４０は、命令実行ユニット２１０の送出する論理アドレス203を物理アドレスに変換しオペランド用キャッシュメモリ２５０に送る。命令実行ユニットは、命令フェッチユニット２００より受取つた命令をデコードし、その結果に従つて、オペランドのアドレス計算，オペランドフェッチ、及び演算を行なう。共有バスモニタ２６０は、共有バス１０２上のトランザクションを監視し、必要ならば、オペランド用キャッシュメモリ２５０の無効化，更新等を行なう。これによつて、複数の演算処理装置に設けられたオペランドキャッシュメモリの一致保証が行なわれる。

次に第４図を用いて命令フェッチユニット200の詳細について説明する。フェッチポインタ300はフェッチすべき命令のアドレスを保持する。命令が逐次的にフェッチされる限り、セレクタ302は加算器３０１を選択しており、フェッチポインタは特定の増分を加算される。本実施例では、１回の命令フェッチで読出すデータ幅を１６Bytesとしているため、フェッチポインタ３００の増分は１６となつている。フェッチした命令の内に分岐命令が存在した場合、セレクタ３０２は、分岐予測バッファ３３０、あるいは、命令実行ユニットより送られてくる分岐先アドレス３０４を選択し、分岐先アドレスをフェッチポインタ３００にセットする。フェッチポインタ３００のアドレスに従つて、命令用キャッシュメモリ２３０より読出された命令は、セレクタ３０３を介して、命令バッファ３１０に格納される。命令バッファ310は、ファーストイン・ファーストアウト方式のバッファであり、そのサイズは、ここでは、１６Bytes×８ エントリで考える。３１２は、命令バッファ３１０の読出しアドレスレジスタである。読出しアドレスレジスタ３１２は、命令バッファ３１０の任意のバイト位置を示しており、アライナ３１１は、該バイト位置より１６Bytes の情報を読出して、デコーダ３１４に送る。命令切出し部３１５より、切出した命令のサイズが加算器３１３に転送され、新たな読出しアドレスレジスタ３１２の値が決定される。デコーダ部３１４は、命令バッファから読出された１６Bytes の情報を、命令の最小単位と等しいビット幅を持つ複数のデコーダによつてデコードを行なう。ここでは、命令の最小単位を２Bytes としており、１６Bytes の情報は、２Bytes 毎８つのデコーダで同時にデコードしている。これら８つのデコーダの分析結果は、命令切出し部３１５に転送される。命令切出し部３１５は、デコーダ部の情報に従い、第１命令３１９の切出し、及び、第１命令のサイズ３１６の識別、並びに第２命令３２５の切出し、及び第２命令のサイズ３２５の識別を行なう。こ

の例では、同時に２命令の切出しとしたが、当然のことながら２命令以上の切出しを同時に行なう構成も可能である。以上の様なデコード方式を採用すれば、可変長命令をデコードする場合でも、同時に複数の命令を切出すことが可能となる。切出された第１命令３１９，第２命令３２５、及び、それぞれのサイズ情報３１６，３１７は、実行ユニット用命令バッファ３４０に同時に格納される。

一方、プログラムカウンタ３２０は、デコーダ部３１４にて切出される第１命令の主メモリ上のアドレスを保持している。第２命令のアドレスは、プログラムカウンタ３２０に第１命令のサイズ３１６を加算器３２３で加算することによつて求められる。第１，第２命令の主メモリ上のアドレスは、各命令が実行ユニット用命令バッファ340に格納される時に、付加情報として同時に格納される。プログラムカウンタ３２０の更新は、命令の分岐が無い限り命令切出し部３１５より送出される第１，第２命令のサイズの和３１８を、加算器３２１により加算することによつて新たな値が求められる。分岐命令によつてプログラムの流れが変わつた場合には、セレクタ３２２により、分岐予測バッファ３３０からの予測アドレス、あるいは、命令実行ユニットからの分岐先アドレス３２５を選択してプログラムカウンタ３２０にセットする。

次に分岐予測バッファ３３０に関して説明する。分岐予測バッファ３３０に格納される情報は、以下に示す５項目である。

1）有効ビット３３１…各エントリーが有効であることを示す。

2）比較用アドレスタグ３３２…分岐命令のアドレスの一部を保持し、これを外部からのアドレスと比較器３３６で比較することにより、該当する分岐命令が、分岐予測バッファ３３０内に存在するかどうかの検証が行なわれる。

3）分岐予測ビット３３３…該当する分岐命令が、条件分岐命令であつた場合、分岐するか否かの情報を与える。

4）分岐先命令アドレス３３４…該当する分岐命令が、分岐すると予測される分岐先命令のアドレス。

5）分岐先命令３３５…該当する分岐命令が、分岐すると予測される分岐先の命令そのもの。

分岐予測バッファ３３０は、分岐命令が実行された時の履歴を記憶しておき、再度同一の分岐命令が表われた時に、その分岐先を予測するものである。その動作を以下に示す。命令切出し部315において、切出された命令が分岐命令であつた場合、それが、第１命令か第２命令かに基づいてセレクタ３２４を制御し、該分岐命令のアドレスを分岐予測バッファ３３０に送る。分岐予測バッファでは、送られてきたアドレスで、分岐予測バッファの特定のエントリを選択するとともに、比較用アドレスタグ３３２を比較器３３６で比較することにより、該当する分岐命令が登録されているか検証する。該分岐命令が登録されており、且つ、分岐予測ビット３３３が、分岐を示していれば、分岐先アドレス３３４を、プログラムカウンタ３２０及び、フェッチポインタ３００にセットする。この時、命令バッファ３１０は全てクリアされる。次に、分岐先命令３３５をセレクタ３０３を介して命令バッファ３１０に格納する。一方、分岐予測ビット３３３が分岐しないことを示していれば、ノーオペレーションとする。

次に第５図を用いて、命令実行ユニット２１０の詳細を説明する。第５図の例では、２命令を同時に実行する構成を示しているが、２命令以上を同時に実行する構成も容易に実現できる。実行ユニット用命令バッファ３４０より２つの命令が同時に読出され、デコーダ４００，４０１により命令の種類及び、オペランドの種類等が識別される。ここで、デコーダ４００において、先に実行されるべき命令（第１命令）がデコードされ、デコーダ４０１において後に実行されるべき命令（第２命令）がデコードされるものとする。これらの情報は、命令結合判定部４０２に送られる。命令結合判定部では、命令の種類及びオペランドの競合などが検証され、実行ユニット用命令バッファ３４０より読出された２つの命令が結合可能かど

うかの判定を行なう。結合可能な命令の種類は、第７図に示される。ほとんどの命令の組が結合可能であるが、ビットフィールド命令，十進演算命令等は、他の命令と結合できない。又、分岐命令どうし、サブルーチンリンク命令どうしも結合できない。又、一方の命令のデスティネーションオペランドが、他方の命令のソースオペランドとなっている場合には、該２つの命令は結合できない。

命令結合判定部４０２において、結合可能と判定された命令の組は、以降のアドレス計算，オペランドフェッチ，演算の各パイプラインステージで同期して実行される。結合不可と判定された場合には、第１命令のみが、以降のステージに渡され、残った命令は次の第１命令となり、更にその次の命令とともにデコードされ結合判定が行なわれる。

結合された各命令のデコード結果は、第１命令が、レジスタ群４１０－４１４に、第２命令がレジスタ群４１５－４１９にセットされる。レジスタオペランドアドレスレジスタ４１０，４１１は、それぞれ、第１命令のソース，デスティネーションオペランドのレジスタアドレスが格納される。第２命令に関しては、４１８，４１９のレジスタが同等の機能を持つ。又、第１命令がメモリオペランドを含む場合には、ベースレジスタのレジスタアドレスが４１４に、インデクスレジスタのレジスタアドレスが４１３に、デスプレースメント情報が４１２に格納される。第２命令に関しては、レジスタ４１５，４１６，４１７が同等の機能を有する。

次にアドレス計算ステージの処理について説明する。第１命令が、メモリオペランドを含む場合、その論理アドレスを計算する必要がある。メモリオペランドのアドレスは、レジスタ４１４によって指定されるアドレス用レジスタファイル４２０内のベースレジスタの内容と、レジスタ４１３によって指定されるアドレス用レジスタファイル４２０内のインデクスレジスタの内容と、ディスプレースメント情報４１２を、加算器４２１で加算することによって求められ、論理アドレスレジスタ４２５に格納される。第２命令に関しても同様な処理が行なわれ、論理アドレスが、レジスタ４２６に格納される。ここで、アドレス用レジスタファイル４２０は、第１命令，第２命令により共有されており、複数の読出しポートを持つことにより、第１，第２命令で同時にアドレス計算を行なうことが可能となっている。

次に、メモリオペランドの読出しステージについて説明する。第１オペランドが、メモリオペランドを含む場合、アドレス計算ステージによって得られた論理アドレス４２５によってメモリアクセスを行なう。論理アドレス４２５は、オペランド用アドレス変換バッファ４３０により、主メモリ上の物理アドレスに変換される。該物理アドレスによりオペランド用キャッシュメモリ４３１がアクセスされ、読出されたメモリオペランドがレジスタ４３４に格納される。第２命令についても同様にしてレジスタ４３５にメモリオペランドが格納される。オペランド用アドレス変換バッファ４３０、及びオペランド用キャッシュメモリ431は、第１，第２命令によって共有されており、複数の読出しポートを持つことにより、第１，第２命令のメモリオペランド読出しを同時に実行できる。ここで注意すべきこととしては、例えば、第１命令でキャッシュミスヒットが発生し、次のステージに移れない場合には、第２命令も同様に次のステージに移れない。

次にオペランドフェッチステージの構成について説明する。第１命令においてオペランドがレジスタである場合には、レジスタオペランドアドレスレジスタ４３２，４３３の情報に従って、データ用レジスタファイル４４０より、オペランドを読出す。一方、メモリオペランドであれば、レジスタ４３４よりアライナ４４１を経て、オペランドを得る。

又、同一のパイプラインにおいて１つ前に実行された命令の結果をソースオペランドとする場合には、パイプ内バイパスルート４６０によりオペランドを得る。一方、他方のパイプラインにおいて、１つ前に実行された命令の結果をソースオペ

ランドとする場合には、パイプ間バイパスルート４６１によりオペランドを得る。以上の処理は、第２命令においても同様に行なわれる。

演算器４５４，４５５においては、オペランドフエツチステージにより得られたオペランドに対して演算が実行され、その結果が、レジスタ456，４５７に格納される。その後、演算結果は、アドレス用レジスタファイル４２０，データ用レジスタファイル４４０、あるいは、オペランド用キヤツシユメモリ４３１に格納される。演算器４５４，４５５における演算結果の状態（ＺＥＲＯ，over Flov等）４６２，４６３は、状態コード生成回路４５８に転送され、状態レジスタ４５９に反映される。

次に、第６図を用いて、状態コード生成回路について詳細に説明する。状態コード生成回路458は、２つの機能を持つ。１つは、第１命令の演算結果の状態と、第２命令の演算結果の状態を、命令の順序を考慮してマージし、状態レジスタに反映する機能。第２の機能は、条件分岐命令と、条件を生成する命令を同時処理するための条件判定機能である。まず第１の機能については、第１命令の演算器４５４より出力される演算結果の状態４６２、及び第２命令の演算器４５５より出力される状態４６３を状態生成部９１８に入力する。状態生成部９１８では、第１命令が、第２命令よりも先に実行されるべき命令であることを考慮し、第１命令からの状態４６２の上に第２命令からの状態を反映した後、状態レジスタ４５９に格納する。

次に、第２の機能について説明する。条件分岐命令が、第２命令のパイプラインで実行されると仮定すると、分岐条件判定情報がレジスタ９０４に、又、分岐予測バツフアによる予測結果が、レジスタ９０５に格納される。今、分岐条件を生成する命令と条件分岐命令が逐次的に実行されたとすると、条件分岐命令を実行する時点では、分岐条件は既に状態レジスタに反映されている。従つてセレクタ９１４で状態レジスタ４５９を選択し、分岐判定回路９１５に入力する。分岐判定回路９１５は、状態レジスタ４５９と、分岐条件判定情報９０４より分岐するか否かを決定する。その結果と、分岐予測結果９０５を比較器９１６で比較し、一致すれば、ノーオペレーション，不一致であれば、全てのパイプラインをキヤンセルして、正しい方向に分岐する。一方、分岐条件を生成する命令と条件分岐命令を同時に実行する場合を考える。分岐条件生成命令を第１命令、条件分岐命令を第２命令とする。この時、条件分岐命令を実行する時点では、分岐条件は、状態レジスタ459に反映されていない。従つて、この場合には、セレクタ９１４が、第１命令の演算結果の状態462を選択し、分岐判定回路９１５に入力する。分岐判定回路９１５は、第１命令の演算結果の状態４６２と、分岐条件判定情報９０４より分岐するか否かを決定する。その結果と、分岐予測結果９０５と比較器９１６で比較し、一致すればノーオペレーション，不一致であれば、全てのパイプラインをキヤンセルして正しい方向に分岐する。

以上の様な状態コード生成回路４５８の機能により、状態レジスタ４５９が、命令の順序に従つて更新されることを保証し、又、条件分岐命令と、分岐条件を生成する命令を同時に実行することが可能となる。

次に、第１図及び、第８，第９図を用いて、パイプライン動作を説明する。第１図は、２命令の同時処理を実現する本実施例のパイプライン構成を示したものである。命令フエツチステージ500は、命令用キヤツシユメモリ５２０から、複数の命令を同時に読出す。プリデコードステージ501は同時に複数の命令の切出しを行ない、分岐命令があれば、分岐予測バツフア５２１をアクセスして、分岐方向を決定する。命令バツフアステージ５０２では、実行ユニツト用命令バツフアからの命令の読出しを行なう。デコード・アンド・コンバインステージ５０３では、２命令を同時にデコードし、その結果に基づいて、結合可能かどうかの判定を行なう。アドレス計算ステージ５０４，５１１はメモリオペランドの論理アドレスを計算する。アドレス変換ステージ５０６，５１２では、

メモリオペランドの論理アドレスに変換する．オペランドフエツチステージ５０８，５１５では、オペランド用キヤツシユメモリ５２３、あるいは、レジスタファイル５２２よりオペランドを読出す．演算ステージ５０９，５１６では、読出したオペランドに対して演算を行なう．ライトステージ５１０，５１７では、演算した結果を、オペランド用キヤツシユメモリ５２３、あるいはレジスタファイル５２２に格納する．アドレス計算ステージ以降は、同一論理の２本のパイプラインによつて構成さている．デコード・アンド・コンバインステージ５０３において結合された命令はこの２本のパイプラインで同期して実行される．

第８図は、２命令の同時処理が、効率良く実行されている場合のパイプラインステージフローを示している．この中で、３番目，４番目の命令の同時実行は、前述した様に、パイプライン間で、演算結果の状態を転送することによつて実現される．又、９番目と１０番目の命令の並列実行は、９番目のサブルーチンジヤンプ命令に関する分岐予測が成功することによつて実現される．

第９図もまた、２命令同時処理方式におけるパイプラインステージフローを示している．３番目と４番目の命令は、ｄ７のレジスタが競合したために命令の結合ができなかつた例を示している．この場合には、３番目の命令のみが単独で実行され、４番目の命令は、５番目の命令と結合され実行されている．又、８番目と９番目の命令は、結合には成功したが、８番目の命令と、７番目の命令の間で、ａ０レジスタが競合し、８番目の命令が、待たされた場合を示している．この時、８番目の命令と結合に成功した９番目の命令も待たされてしまう．このパイプライン間の同期によつて、命令の逐次性が維持される．

〔発明の効果〕

本発明によれば、命令の逐次性を維持して複数命令を同時に実行できるので、汎用レジスタファイルの排他制御を簡略化でき高性能化が可能となる．又、唯一の状態レジスタが、命令の順序に従つて更新されることを保証できるので、演算処理装置の状態の管理が容易になる．

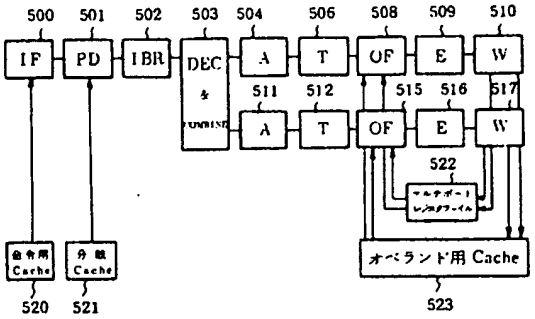
4．図面の簡単な説明

第１図は、本発明の一実施例のパイプライン構成図、第２図は、本発明が適用される計算機システムの構成図、第３図は、第２図における演算処理装置の内部構成、第４図は、第３図における命令フエツチユニツトの内部構成、第５図は、第３図における命令実行ユニツトの内部構成、第６図は、第５図における状態コード生成回路の内部構成、第７図は、結合可能な命令の組合せ図、第８図，第９図は、パイプラインステージフローの一例である．

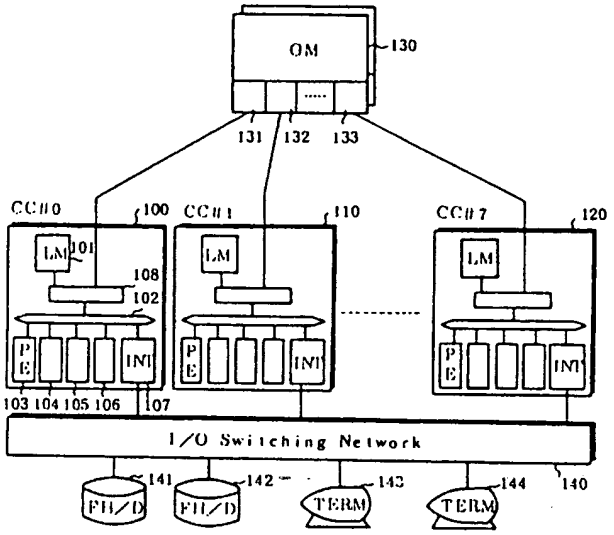
５００…命令フエツチステージ、５０３…デコード・アンド・コンバインステージ、５０４，５１１…アドレス計算ステージ、５０８，５１５…オペランドフエツチステージ、５０９，５１６…演算ステージ、５２２…マルチポートレジスタファイル、５２３…オペランド用キヤツシユ．

代理人　弁理士　小川勝男

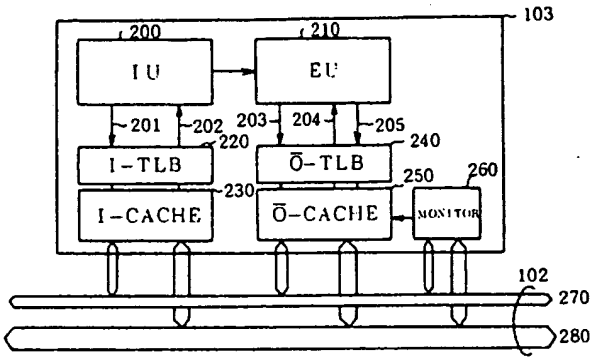
第　１　図



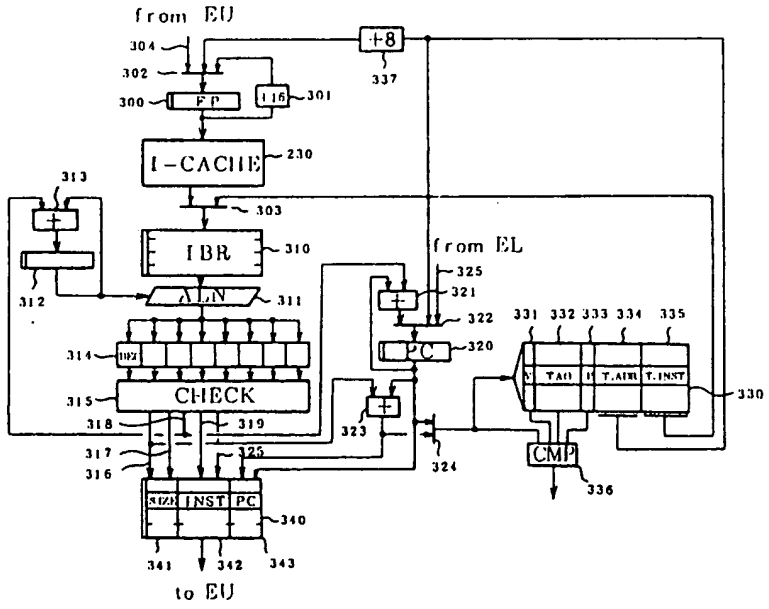
第 2 図


OM
130
131
132
133
CC#0
100
CC#1
110
CC#7
120
LM
101
108
102
PE
INT
103 104 105 106
107
I/O Switching Network
141
142
143
144
140
FD/D
TERM

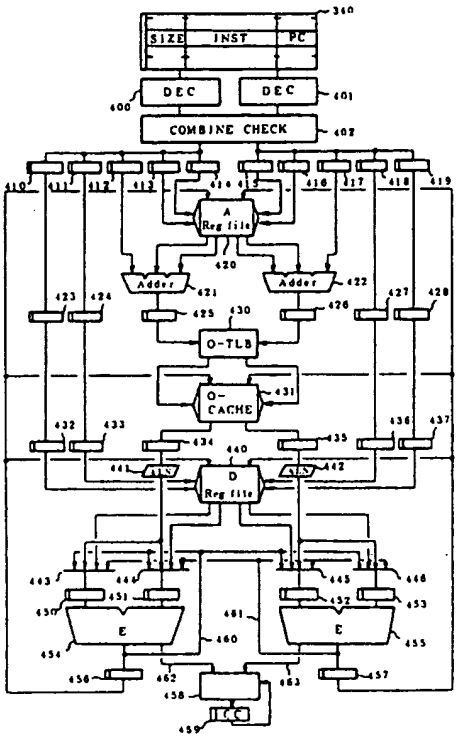
第 3 図


103
200
210
IU
EU
201
202
203
204
205
I-TLB
220
Ō-TLB
240
250
260
I-CACHE
230
Ō-CACHE
MONITOR
102
270
280

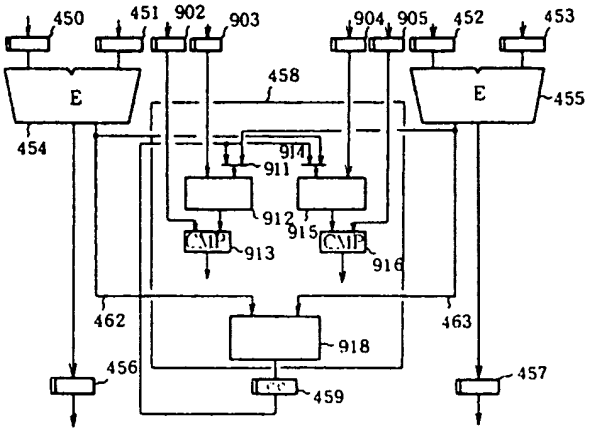
第 4 図


from EU
304
302
300
FP
116
301
+8
337
I-CACHE
230
313
303
IBR
310
from EL
325
321
322
331 332 333 334 335
312
ALN
311
PC
320
314
DEC
CHECK
315
318
319
323
324
317
316
325
CMP
336
330
SIZE
INST
PC
340
341
342
343
to EU

第 5 図



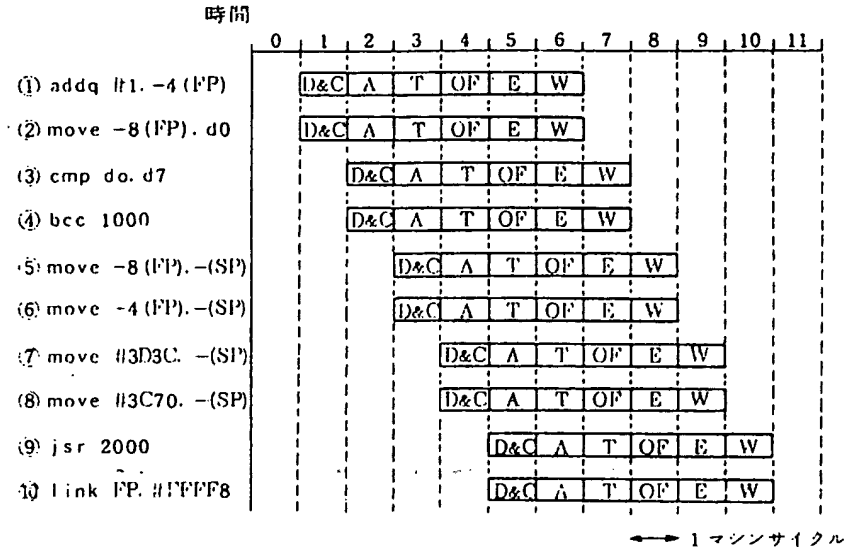
第 6 図



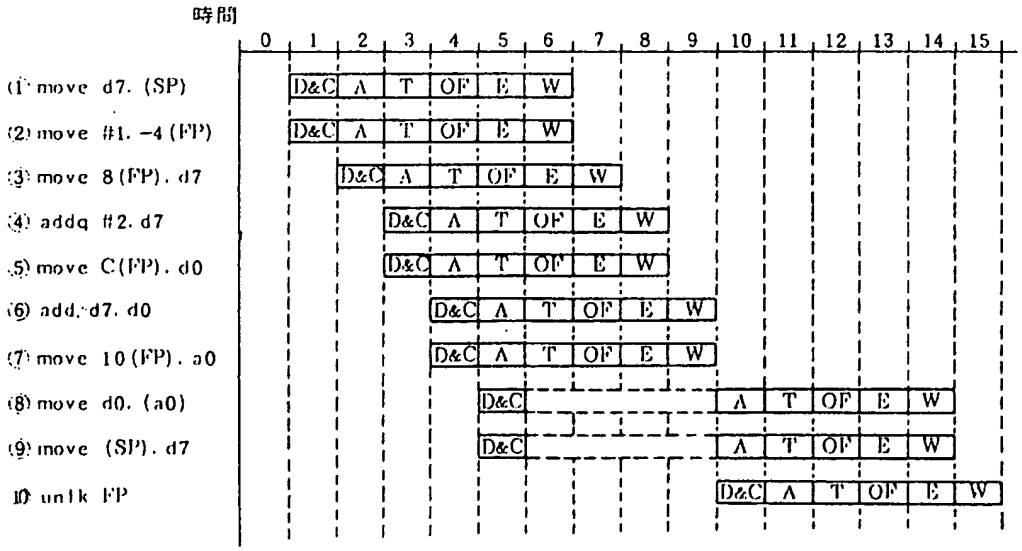
第 7 図

		固定小数点			浮動小数点			分岐	サブルーチンリンク	ビットフィールド命令
		ロード	ストア	演算	ロード	ストア	演算			
固定小数点	ロード	○	○	○	○	○	○	○	○	×
	ストア		○	○	○	○	○	○	○	×
	演算			○	○	○	○	○	○	×
浮動小数点	ロード				○	○	○	○	○	×
	ストア					○	○	○	○	×
	演算						○	○	○	×
分岐								×	○	×
サブルーチンリンク									×	×
ビットフィールド命令										×

第 8 図


時間
0 1 2 3 4 5 6 7 8 9 10 11
(1) addq #1. -4 (FP)
(2) move -8 (FP). d0
(3) cmp do. d7
(4) bcc 1000
(5) move -8 (FP). -(SP)
(6) move -4 (FP). -(SP)
(7) move #3D3C. -(SP)
(8) move #3C70. -(SP)
(9) jsr 2000
(10) link FP. #FFFF8
D&C A T OF E W
1マシンサイクル

第 9 図


時間
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(1) move d7. (SP)
(2) move #1. -4 (FP)
(3) move 8 (FP). d7
(4) addq #2. d7
(5) move C (FP). d0
(6) add. d7. d0
(7) move 10 (FP). a0
(8) move d0. (a0)
(9) move (SP). d7
(10) unlk FP
D&C A T OF E W